Input and Output in LISP

York University CSE 3401

Vida Movahedi

Overview

- Read and Print
- Escape characters in symbol names
- Strings & how to format them
- File I/O

[ref.: Chap. 10 -Wilensky]

Read

- (read)
  - Read can be used a function of no arguments
  - It reads one s-expression from the standard input
  - And returns it.

> (setq x (read))
20 ← input by user
20
> x
20

> (setq y (read))
(a b) ← input by user
(A B)

> y
(A B)

> (setq z (cons 'a (read))
(b c) ← input by user
(A B C)
> z
(A B C)

Print

- (print arg)
  - Print can be used a function with one argument
  - The one argument must be an s-expression
  - It prints to the standard output,
    - A new line
    - Then its argument
    - Then a single space
  - Returns its argument

> (print 'enter)
ƀ ← **new line**
ENTERƀ ← **single space** **PRINTED**
ENTER **RETURNED**

Print (cont.)

- Example:

```
> ((lambda () (print 'enter) (setq x (read))))
                              - blank line
ENTER  10                     - 10 entered by user
10                            - 10 returned by the last form, (setq ...)

> (let () (print 'enter) (print 'a) (print 'number) (setq x (read)))
                              - blank line
ENTER
A
NUMBER 20                     - 20 entered by user
20                            - 20 returned by the last form, (setq ...)
```

Prin1, Terpri

- (prin1 arg)
  - Only prints its argument (no new lines or spaces)
  - Returns its argument
- (terpri)
  - Stands for “terminate print line”
  - Prints a carriage return (new line)
  - Returns NIL

```
> (prog ()
    (prin1 'enter>)
    (if (numberp (read)) (prin1 'ok) (prin1 'Nop!))
    (terpri))
ENTER>11
OK
NIL
```

Part of symbol’s name

Value returned by terpri? No, prog returns NIL when done.

Example (1)

```
> (loop
        (print 'number>)
        (let ((in (read)))
              (if (equal in 'end) (return nil))
                  (print (sqrt in)) ))
ƀ
NUMBER>ƀ25
ƀ
5ƀ
NUMBER>ƀ9
ƀ
3ƀ
NUMBER>ƀend
NIL
```

Example (2)

```
> (loop
        (print  '(A number please>))
         (return  (read)))
ƀ
(A NUMBER PLEASE>)ƀ20
20

> (let ()
        (mapc  'prin1  '(A number please>))
        (read))
ANUMBERPLEASE>10
10
```

Prints the parenthesis!

No spaces!

Escape characters

- Any way to add a space?! YES!
- **Method 1**: Add a space to symbol's name
  - \ (single escape character): allows the character following it to escape the normal LISP interpretation
  - | (multiple escape character): anything between a pair of vertical bars escapes the normal LISP interpretation

(setq ab(c 10) → waits for the closing parenthesis

(setq ab\(c 10) → sets the value of the symbol ab(c

Escape characters (cont.)

```
> (setq  |a var|  10)
10
> |a var|
10

> (setq  |BigVar|  200)
200
> |BigVar|
200

> 'BigVar
BIGVAR
> '|BigVar|
|BigVar|
```

Can have spaces in symbol's name

No changing to UPPER CASE, if escape characters used.

Example (3)

```
> (let ()
      (mapc 'prin1 '(a |ƀnumberƀ| please\ƀ>))
      (read))
A|ƀnumberƀ||PLEASEƀ>|
```

Prints the escape characters!!!

Note PLEASE is in UPPER CASE, but number is not.

```
> (print '|Aƀnumberƀpleaseƀ>|)
ƀ
|A number please >|
|A number please >|
```

princ

- Is there a way to print anything looking nice?! YES ☺ Use **princ**:

```
> (prog () (princ '|A number please> |) (read))
A number please> 100
NIL
```

- The return value of print and princ are the same, only the printed output is different.

```
> (princ  '|A␣number␣please␣>|)
A number please >
|A number please >|
```

- **Princ** does NOT prints **s-expression**, but prints in human-readable format. If what is being printed needs to be read or used by LISP use **print** to print s-expressions

Strings

- Other data types, such as strings have been added to LISP to increase functionality
- A string is a sequence of characters enclosed in double quotes, e.g. “Hello there!”
- **Method 2:** Use strings

> ((lambda () (princ “A number please: “) (read)))
A number please: 100
100

– We still need to use **princ** for not having the double quotes

Strings (cont.)

> “Hello there!”

“Hello there!”

> (print “Hi”)

ƀ

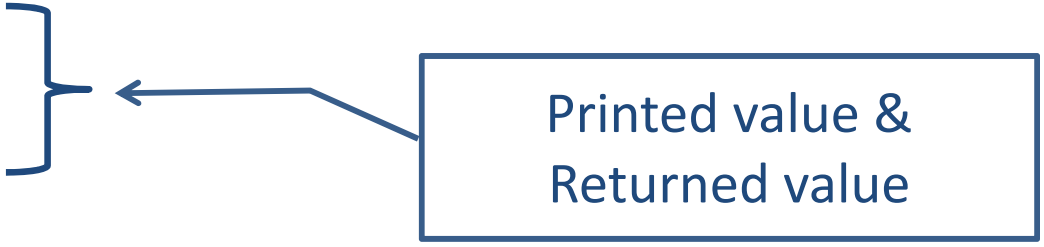
“Hi”

“Hi”

> (princ “Hi”)

Hi

“Hi”



Strings (cont.)

- A symbol's name (also called print name) is a string.

  > (symbol-name 'x)
  "X"

  > (symbol-name 'BigVar)
  "BIGVAR"

  > (symbol-name 'ab\(c)
  "AB(C"

  > (symbol-name '|A Big Var|)
  "A Big Var"

- Strings don't have components (values, property lists, etc), therefore require less storage space

Format

- (format destination string ....)
  - Destination:
    - Nil: just return the formatted string
    - T: to standard output
    - Any other stream
  - String (can contain directives)

    ~A or ~nA Prints one argument as if by PRINC

    ~S or ~nS Prints one argument as if by PRIN1

    ~D or ~nD Prints one argument as a decimal integer

    ~F or ~nF Prints one argument as a float

    ~O,~B, ~X Prints one argument as an octal, binary, or hexidecimal

    ~% Does a TERPRI

    where **n** is the width of the field in which the object is printed

Format (cont.)

> (setq n 32)
32

> (format t "N is ~d" n)
N is 32
NIL

> (format nil "N is ~d" n)
"N is 32“

> (format nil "N is ~5d" n)
“N is 32”

> (format nil "N is ~10b" n)
“N is 100000”

> (format nil "N is ~10,'0b" n)
“N is 0000100000”

> (format nil "N is ~:b" n)
“N is 100,000”

> (format nil "N is ~d~%" n)
"N is 32
“

> (format nil "N is ~7,2f" n)
“N is 32.00”

> (format nil "Hi ~a" "Bob")
"Hi Bob“

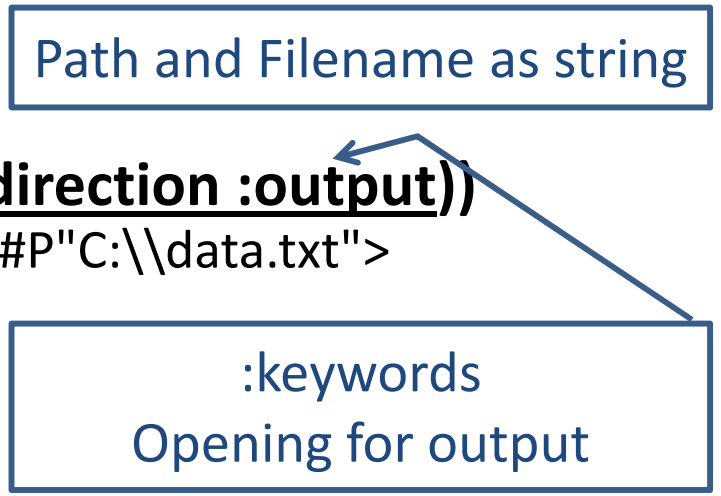
> (format nil "Hi ~s" "Bob")
"Hi \"Bob\"“

> (format nil "Hi ~s" '|Bob|)
"Hi |Bob|“

> (format nil "Hi ~a" '|Bob|)
"Hi Bob"

Files

- Writing to files




```
> (setq  outstream  (open "c:\\data.txt" :direction :output))
#<OUTPUT BUFFERED FILE-STREAM CHARACTER #P"C:\\data.txt">
> (print   '(1 2 3)  outstream)
(1  2  3)
> (close  outstream)
T
```

- Reading from files

```
> (setq  instream  (open "/usr/lisp/file.dat" :direction :input))
#<INPUT BUFFERED FILE-STREAM CHARACTER #P"C:\\lispcode\\file.txt" @1>
> (read instream)
(1  2  3)
> (close instream)
T
```

Files (cont.)

- What happens when reaching end of file?

  > (read instream)
  Error - going beyond end of file!

  > (read instream nil ‘eof)
  EOF

  >(read instream nil ‘oops)
  OOPS

(read stream eof-error-p eof-value)
If eof-error-p is T,
generates error if eof reached.
If eof-error-p is NIL,
returns eof-value if eof reached.

Files (cont.)

- Standard input and output
  - When stream arguments are not supplied to read and print, the standard streams are used.
  - The standard streams are stored in ***standard-input*** and ***standard-output***.
- **Princ** can also be used for writing to files in human-readable format. Not necessarily readable by read.

Dribble

- (dribble pathname)
  Starts recording any interactions with the interpreter

- (dribble)
  Stops recording

  For example:
  > (dribble “c:\\mydribble.txt”)
  > (setq x 10)
  10
  >(setq y (cons ‘a x))
  (A . 10)
  >(dribble) → The above interactions will be saved in the file.

Final notes

- Note that the top-level of LISP (the interpreter) is just a loop that
  - Reads from the standard input
  - Evaluates
  - Prints the returned value to the standard output
  - Referred to as the **read-eval-print loop**
- LISP conatins many other built-in functions for reading characters, reading lines, printing lists, etc that we did not cover.